**Managing Packages with Npm**

- Introduction to managing packages with npm
- Managing Packages with npm Challenges
- How to Use package.json, the Core of Any Node.js Project or npm Package
- Add a Description to Your package.json
- Add Keywords to Your package.json
- Add a License to Your package.json
- Add a Version to Your package.json
- Expand Your Project with External Packages from npm
- Manage npm Dependencies By Understanding Semantic Versioning
- Use the Tilde-Character to Always Use the Latest Patch Version of a Dependency
- Use the Caret-Character to Use the Latest Minor Version of a Dependency

Managing Packages with Npm - How to Use package.json, the Core of Any Node.js Project or npm Package

Introduction
The Node Package Manager (npm) is a command-line tool used by developers in order to control and share packages or modules of JavaScript code written for use with Node.js.

Npm generates a package.json file which lists the dependencies for the project, whenever a new project is started. Npm packages are regularly updated and changes can be done to the package.json file to set specific version numbers for each dependency. This ensures that updates to a package don't break your project.

npm saves packages in a folder named node_modules. These packages can be installed in two ways:

- globally in a root node_modules folder in order to be accessible by all projects.
- locally within a project's own node_modules folder, accessible only to that project.

In order to create a separation between the dependencies of different projects, developers prefer to install packages local to each project.

Each challenge has been completed in Glitch, a starter project and and the public Glitch url has been copied to free code camp challenge screen in order to test it.

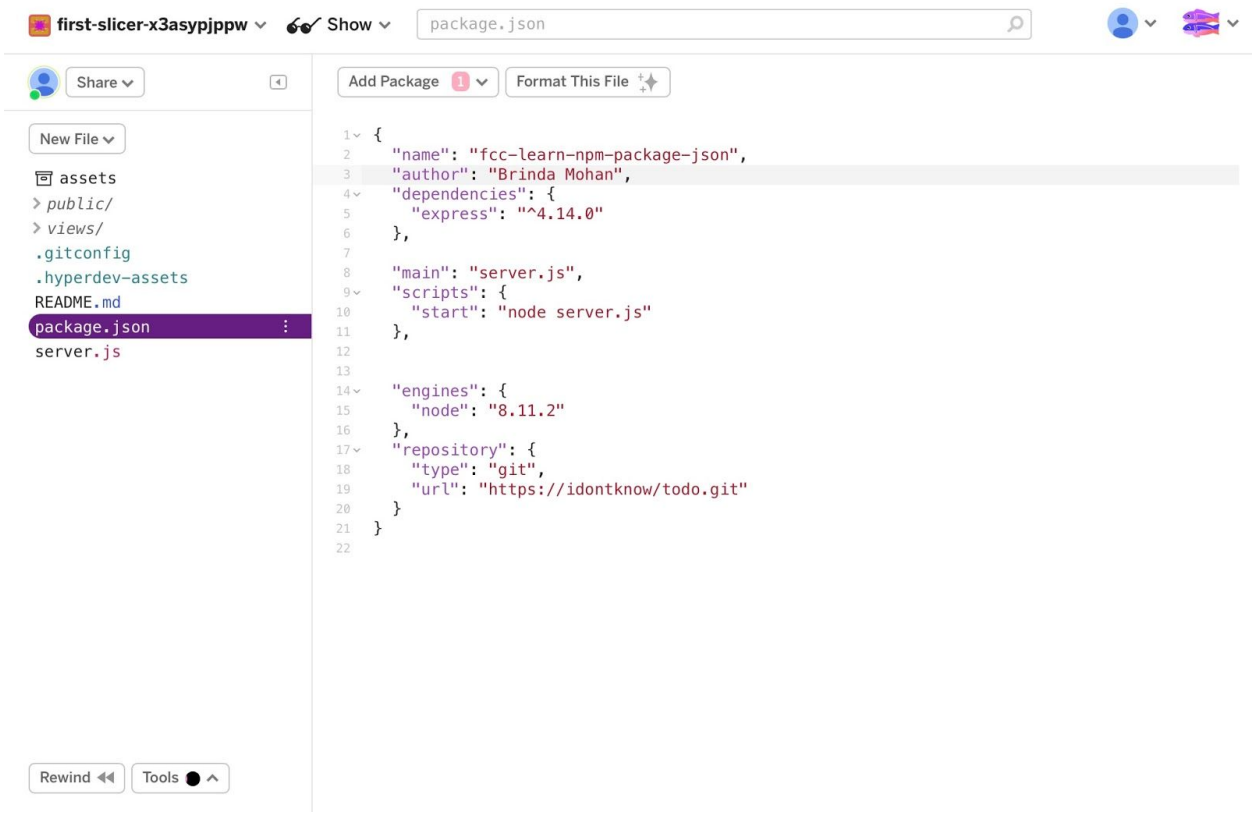
The following project have been completed Glitch tool.

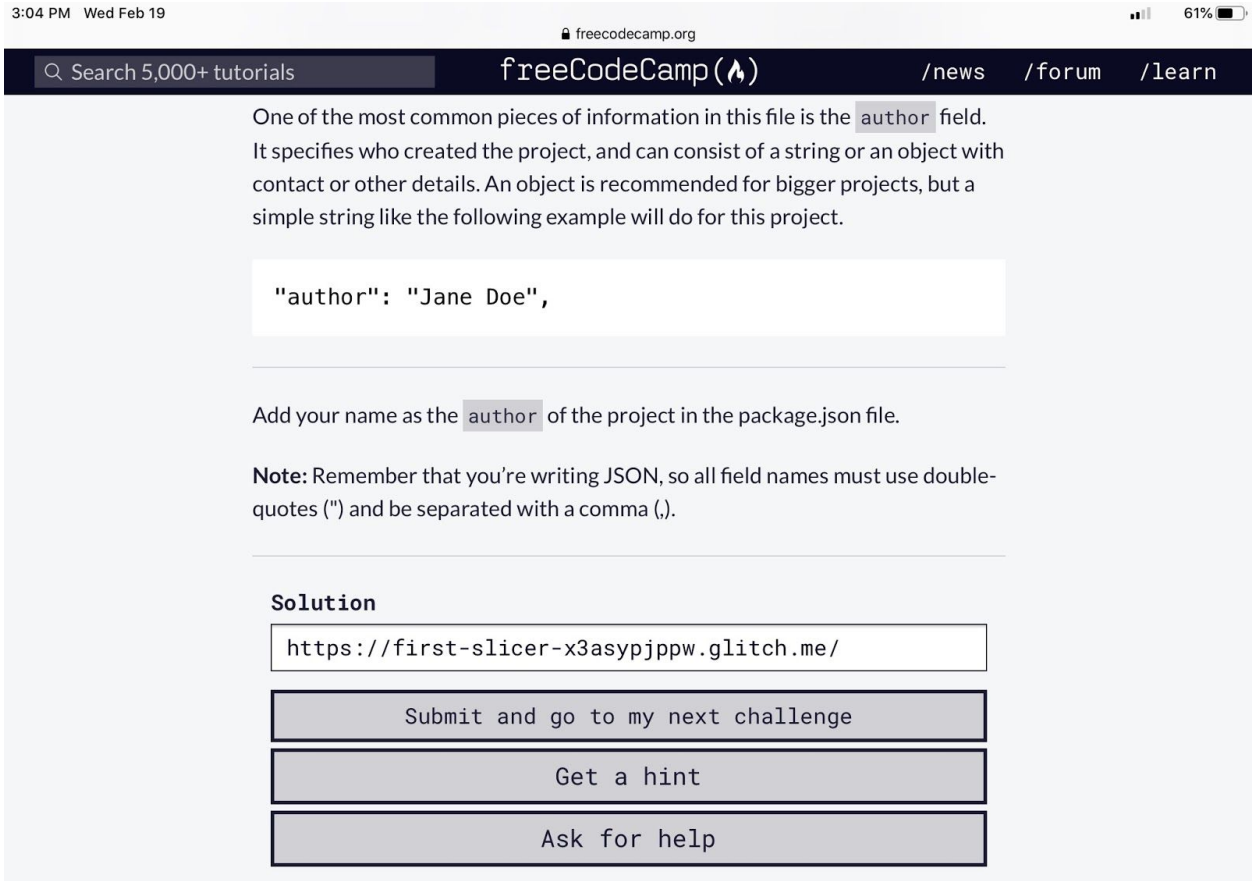
Task1:

The package.json file is the center of any Node.js project or npm package stores information about the project. It consists of a single JSON object where information is stored in key- value pairs. It is similar to the <head> section of an HTML document which describes the content of a webpage. There are only two required fields; "name" and "version". It is a good practice to provide additional information to future users and for maintenance purposes such as description etc.

The author field is one of the most common pieces of information in this file and can consist of a string or an object with contact or other details. For bigger projects, an object is recommended but a simple string is used for this project as in the screen shot below.

Add name as the author of the project in the package.json file.






Task 2:

The description field is the next part of a good package.json file where a short, but informative description about the project belongs. Description field helps other developers and future maintainers to understand the project quickly as it summarizes what a project does.

Managing Packages with Npm - Add a Description to package.json

4:08 PM Wed Feb 19 54%

glitch.com

Vi | Appl | The world'... | Add a Des... | packag... | Backend C... | NOTIFICAT... | Google Co... | JSON Synt... | Favorites | freeCodeC...

first-slicer-x3asypjppw | Show | package.json

Share

New File

- assets
- public/
- views/
- .gitconfig
- .hyperdev-assets
- README.md
- package.json
- server.js

Add Package 1 | Format This File

```
{
  "name": "fcc-learn-npm-package-json",
  "author": "Brinda Mohan",
  "description": "An awesome project",
  "dependencies": {
    "express": "^4.14.0"
  },

  "main": "server.js",
  "scripts": {
    "start": "node server.js"
  },


  "engines": {
    "node": "8.11.2"
  },
  "repository": {
    "type": "git",
    "url": "https://idontknow/todo.git"
  }
}
```

Rewind | Tools


Task 3:
Add an array of suitable strings to the keywords field in the package.json file of the project.

- package.json should have a valid "keywords" key
- "keywords" field should be an Array
- "keywords" should include "freecodecamp"

first-slicer-x3asypjppw Show package.json

Share

New File

- assets
- public/
- views/
- .gitconfig
- .hyperdev-assets
- README.md
- package.json
- server.js

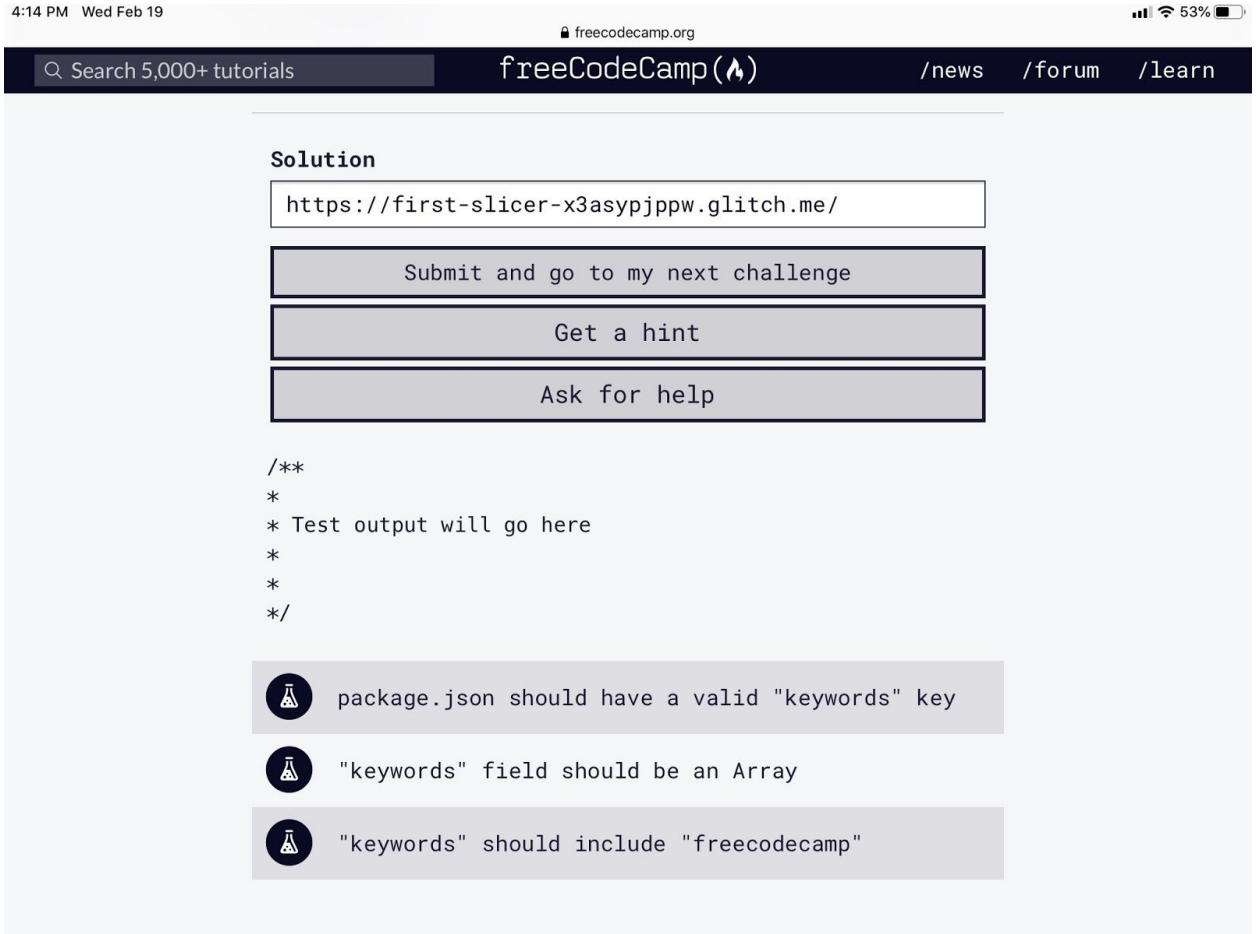
Rewind Tools

Add Package 1 Format This File

```
{
  "name": "fcc-learn-npm-package-json",
  "author": "Brinda Mohan",
  "description": "An awesome project",
  "keywords": [ "descriptive", "related", "words", "freecodecamp" ],
  "dependencies": {
    "express": "^4.14.0"
  },

  "main": "server.js",
  "scripts": {
    "start": "node server.js"
  },


  "engines": {
    "node": "8.11.2"
  },
  "repository": {
    "type": "git",
    "url": "https://idontknow/todo.git"
  }
}
```




Task 4:
Managing Packages with Npm - Add a License to package.json

The license field informs users of what they are allowed to do with your project.

Some common licenses for open source projects include MIT and BSD. Although License information is not required, and copyright laws in most countries will give the developer ownership of what was created by default, it’s always a good practice to explicitly state what users are allowed to do. Here's an example of the license field:

- package.json should have a valid "license" key

first-slicer-x3asypjppw Show package.json

Share

New File

assets
public/
views/
.gitconfig
.hyperdev-assets
README.md
package.json
server.js

Rewind Tools

Add Package 1 Format This File

```
{
  "name": "fcc-learn-npm-package-json",
  "author": "Brinda Mohan",
  "description": "An awesome project",
  "keywords": [ "descriptive", "related", "words", "freecodecamp" ],
  "license": "MIT",
  "dependencies": {
    "express": "^4.14.0"
  },

  "main": "server.js",
  "scripts": {
    "start": "node server.js"
  },


  "engines": {
    "node": "8.11.2"
  },
  "repository": {
    "type": "git",
    "url": "https://idontknow/todo.git"
  }
}
```


Task 5:
Managing Packages with Npm - Add a Version to package.json

- package.json should have a valid "version" key

first-slicer-x3asypjppw Show package.json

Share

New File

assets
public/
views/
.gitconfig
.hyperdev-assets
README.md
package.json
server.js

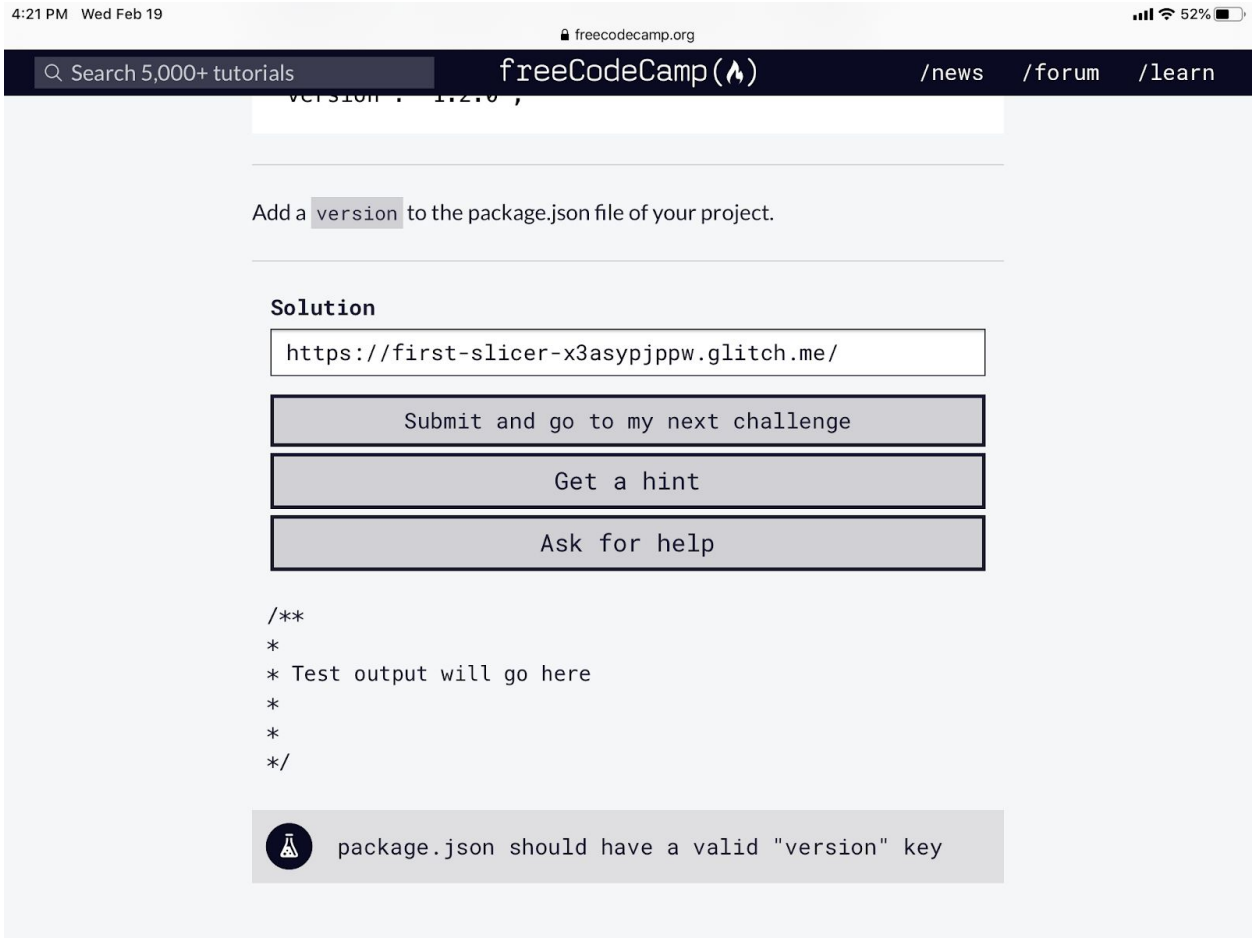
Rewind Tools

Add Package 1 Format This File

```
{
  "name": "fcc-learn-npm-package-json",
  "author": "Brinda Mohan",
  "description": "An awesome project",
  "keywords": [ "descriptive", "related", "words", "freecodecamp" ],
  "license": "MIT",
  "version": "1.3.0",
  "dependencies": {
    "express": "^4.14.0"
  },

  "main": "server.js",
  "scripts": {
    "start": "node server.js"
  },


  "engines": {
    "node": "8.11.2"
  },
  "repository": {
    "type": "git",
    "url": "https://idontknow/todo.git"
  }
}
```




Task 6
Managing Packages with Npm - Expand Your Project with External Packages from npm

- "dependencies" should include "moment"
- "moment" version should be "2.14.0"

4:28 PM Wed Feb 19 51%

glitch.com

The wor | Favorites | Vivekacho... | Expand Yo... | packag... | Backend C... | NOTIFICAT... | Google Co... | JSON Synt... | Managing...

first-slicer-x3asypjppw | Show | package.json

Share | Add Package 1 | Format This File

New File

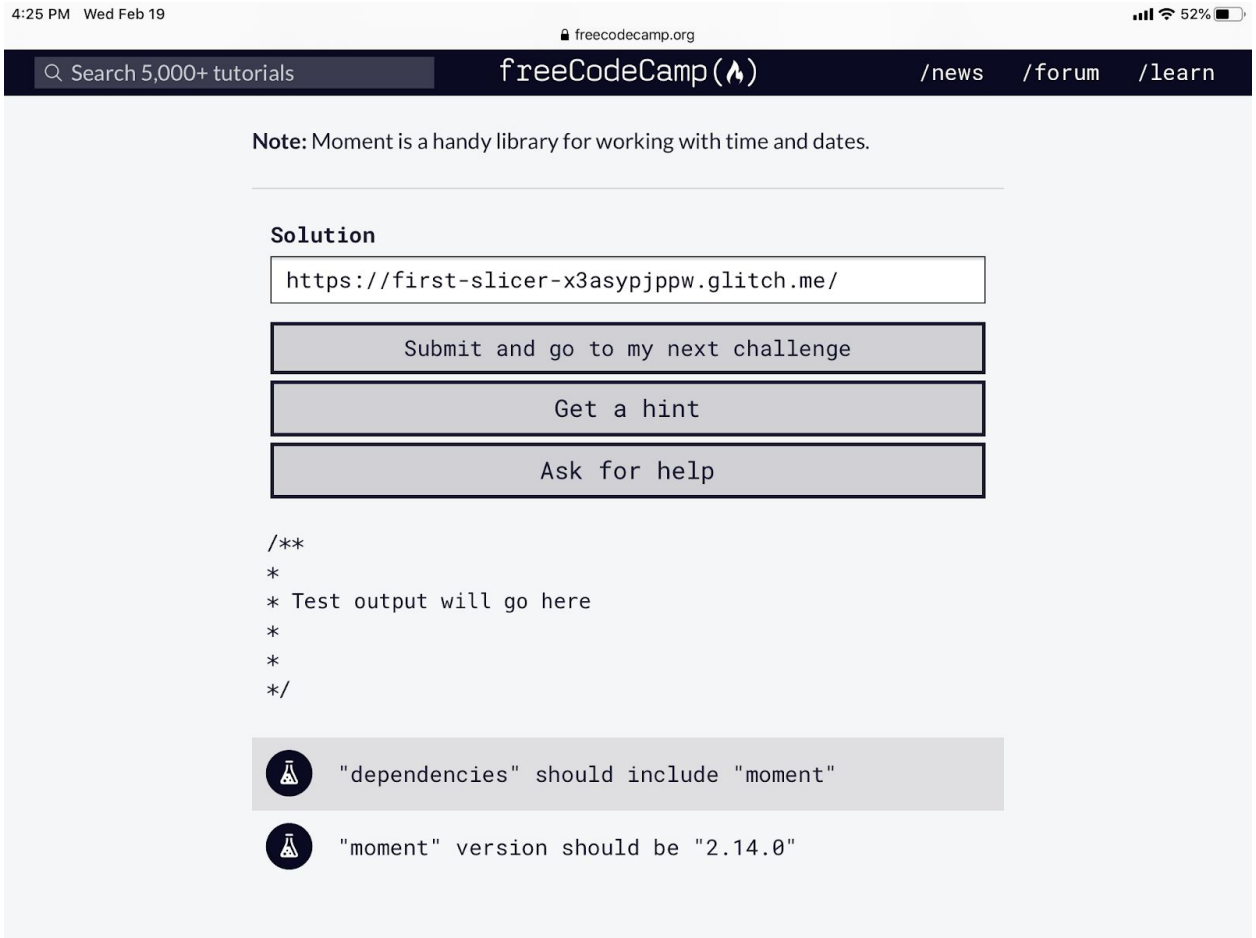
- assets
- public/
- views/
- .gitconfig
- .hyperdev-assets
- README.md
- package.json
- server.js

```
{
  "name": "fcc-learn-npm-package-json",
  "author": "Brinda Mohan",
  "description": "An awesome project",
  "keywords": [ "descriptive", "related", "words", "freecodecamp" ],
  "license": "MIT",
  "version": "1.3.0",
  "dependencies": {
     "package-name": "version",
     "moment": "2.14.0"
  },

  "main": "server.js",
  "scripts": {
    "start": "node server.js"
  },


  "engines": {
    "node": "8.11.2"
  },
  "repository": {
    "type": "git",
    "url": "https://idontknow/todo.git"
  }
}
```

Rewind | Tools




Task 7:
Managing Packages with Npm - Manage npm Dependencies By Understanding Semantic Versioning

In the dependencies section of your package.json file, versions of the Npm packages follow what's called Semantic Versioning (SemVer). SemVer is an industry standard for software versioning aiming to make it easier to manage dependencies.

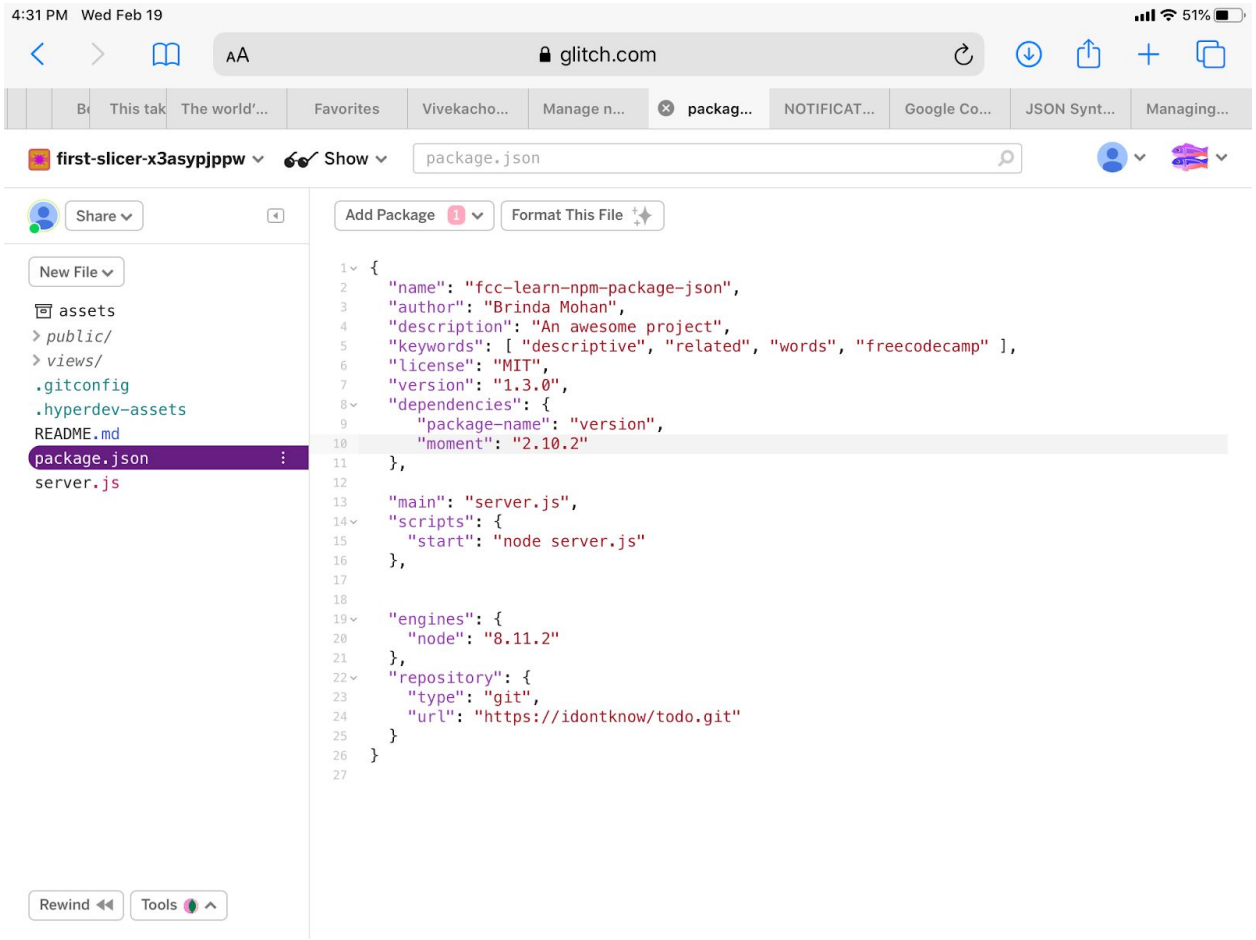
SemVer must be used by libraries, frameworks or other tools published on Npm in order to clearly communicate what kind of changes projects can expect if they update.

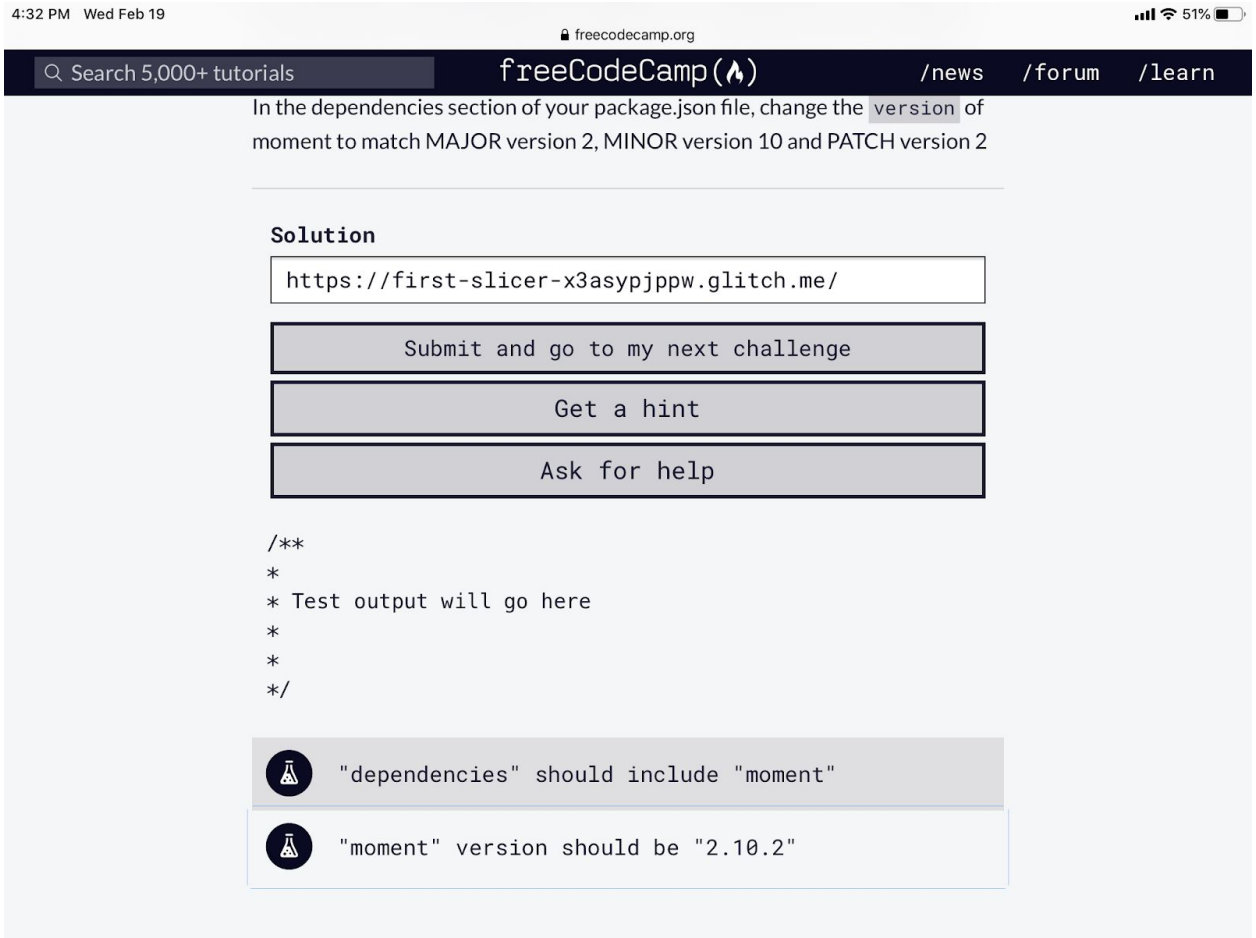
Understanding SemVer can be useful when developing software that uses external dependencies. Understanding the numbers will save from accidentally introducing breaking changes to the project without understanding why things that worked yesterday suddenly don't work today. According to the official website Semantic Versioning works as follows:

"package": "MAJOR.MINOR.PATCH"

The MAJOR version should increment when you make incompatible API changes. The MINOR version should increment when you add functionality in a backwards-compatible manner. The PATCH version should increment when you make backwards-compatible bug fixes. PATCHes are bug fixes and MINORs add new features but neither of them break what worked before. MAJORs add changes that won't work with earlier versions.

- In the dependencies section of your package.json file, change the version of moment to match MAJOR version 2, MINOR version 10 and PATCH version 2






Task 8

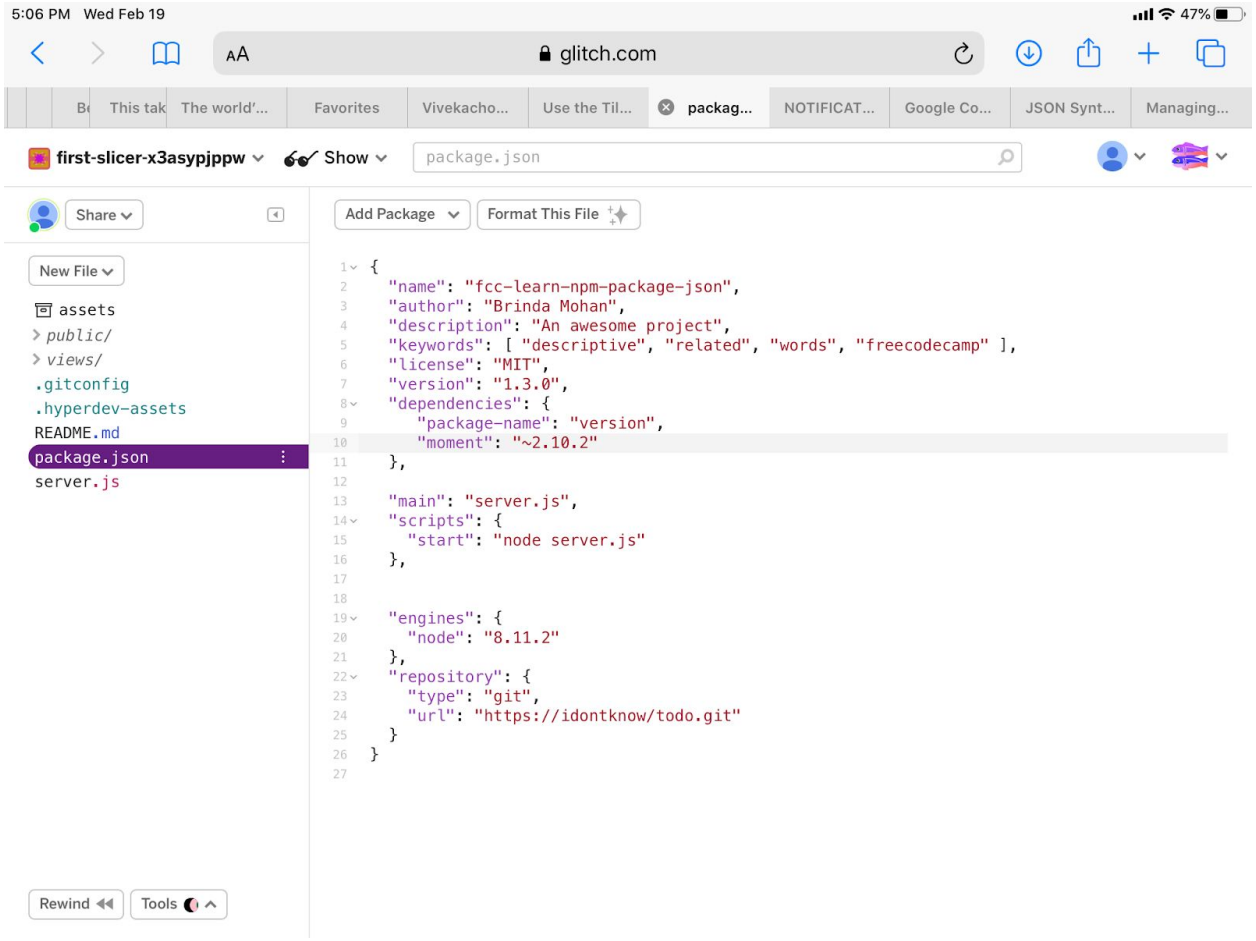
Managing Packages with Npm - Use the Tilde-Character to Always Use the Latest Patch Version of a Dependency

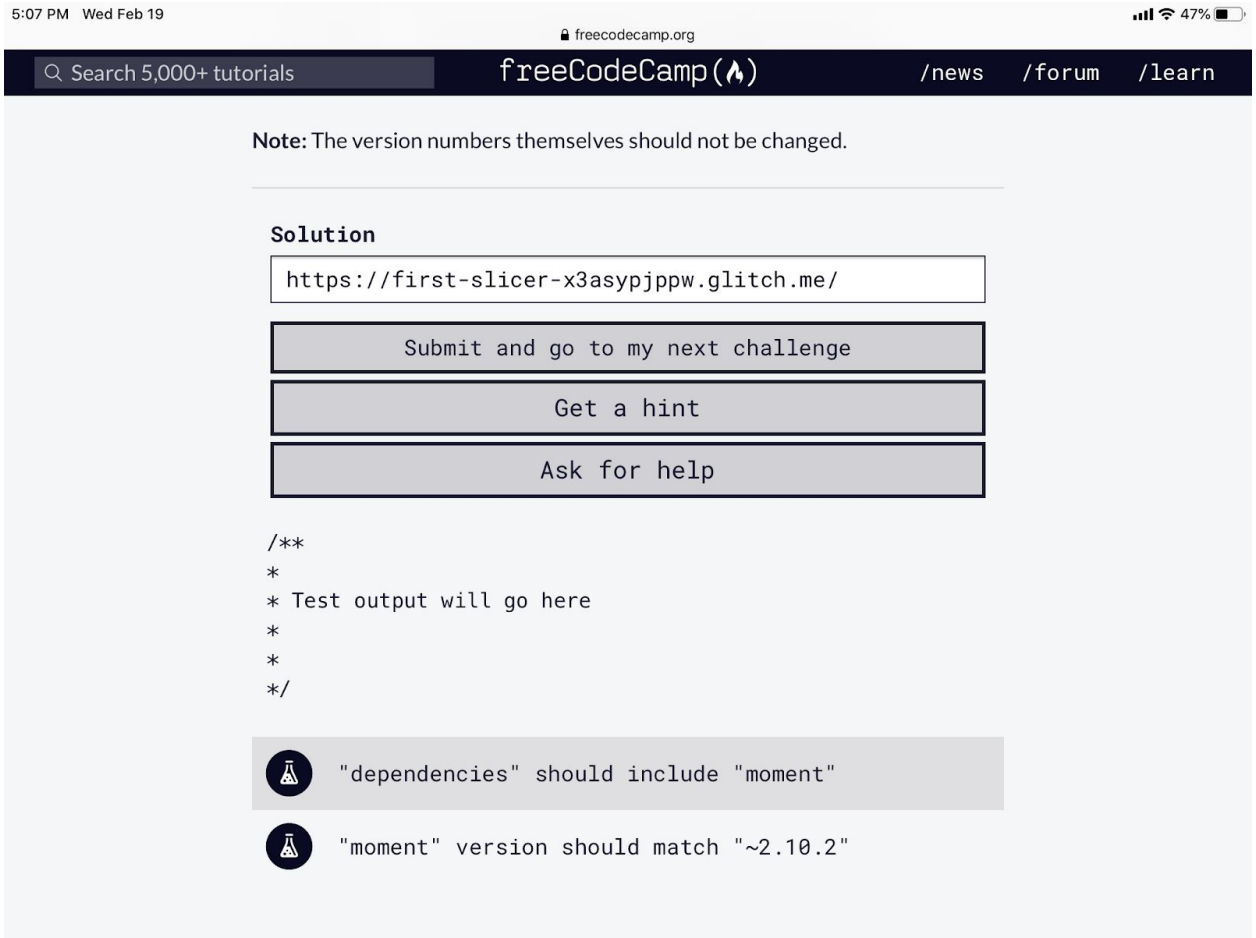
In the last challenge, you told npm to only include a specific version of a package. That's a useful way to freeze your dependencies if you need to make sure that different parts of your project stay compatible with each other. But in most use cases, you don't want to miss bug fixes since they often include important security patches and (hopefully) don't break things in doing so.

Npm dependency to update to the latest PATCH version, can be done by adding a prefix to the dependency's version with the tilde (~) character. For example, "package": "~1.3.8"

Use the tilde (~) character to prefix the version of moment in your dependencies, and allow npm to update it to any new PATCH release.
The version numbers themselves should not be changed.

- "dependencies" should include "moment"
- "moment" version should match "~2.10.2"






Task 9

Managing Packages with Npm - Use the Caret-Character to Use the Latest Minor Version of a Dependency

Installation of future updates can be done to Npm through the caret (^).
Caret will allow both MINOR updates and PATCHes.

For example:
"~2.10.2" - allows npm to install to the latest 2.10.x version.
Using caret (^) as a version prefix, npm would be allowed to update to any 2.x.x version for example:"package": "^1.3.8"
This would allow updates to any 1.x.x version of the package.

Use the caret (^) to prefix the version of moment in your dependencies and allow npm to update it to any new MINOR release.

Note: The version numbers themselves should not be changed.

first-slicer-x3asypjppw Show package.json

Share

New File

assets
public/
views/
.gitconfig
.hyperdev-assets
README.md
package.json
server.js

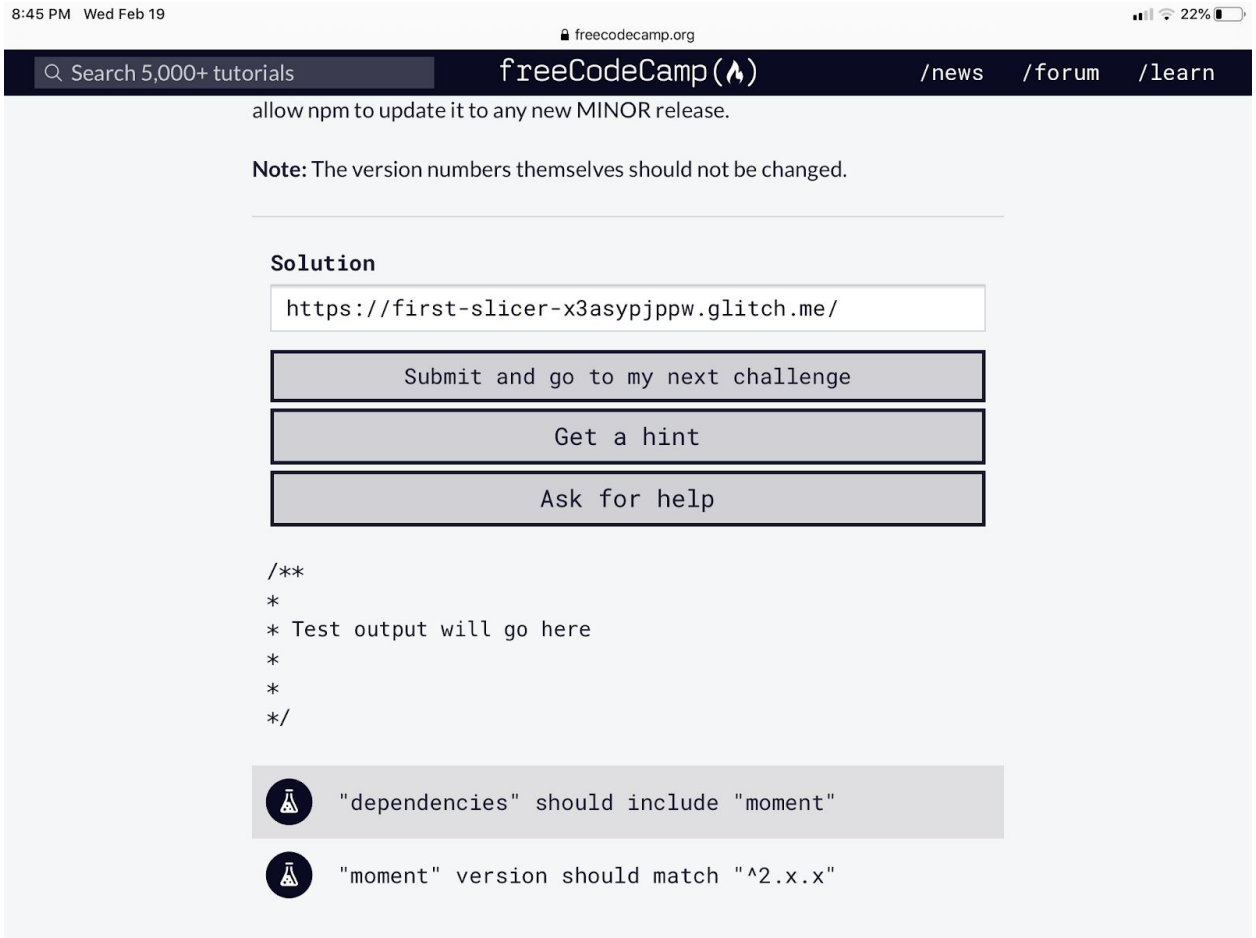
Add Package Format This File

```
{
  "name": "fcc-learn-npm-package-json",
  "author": "Brinda Mohan",
  "description": "An awesome project",
  "keywords": [ "descriptive", "related", "words", "freecodecamp" ],
  "license": "MIT",
  "version": "1.3.0",
  "dependencies": {
     "package-name": "version",
     "moment": "^2.x.x"
  },

  "main": "server.js",
  "scripts": {
    "start": "node server.js"
  },


  "engines": {
    "node": "8.11.2"
  },
  "repository": {
    "type": "git",
    "url": "https://idontknow/todo.git"
  }
}
```

Rewind Tools




Task 10
Managing Packages with Npm - Remove a Package from Your Dependencies

Dependencies for a package can be removed by removing the corresponding key-value pair for that package.

Remove the moment package from dependencies.

Note: Ensure that the right amount of commas are present after the dependencies are removed.

- dependencies" should not include "moment"

first-slicer-x3asypjppw Show package.json

Share

New File

assets
public/
views/
.gitconfig
.hyperdev-assets
README.md
package.json
server.js

Rewind Tools

Add Package Format This File

```
{
  "name": "fcc-learn-npm-package-json",
  "author": "Brinda Mohan",
  "description": "An awesome project",
  "keywords": [ "descriptive", "related", "words", "freecodecamp" ],
  "license": "MIT",
  "version": "1.3.0",
  "dependencies": {
     "package-name": "version"
  },

  "main": "server.js",
  "scripts": {
    "start": "node server.js"
  },


  "engines": {
    "node": "8.11.2"
  },
  "repository": {
    "type": "git",
    "url": "https://idontknow/todo.git"
  }
}
```

**Note:** Make sure you have the right amount of commas after removing it.

**Solution**

https://first-slicer-x3asypjppw.glitch.me/

Submit and go to my next challenge

Get a hint

Ask for help

```
/**
*
* Test output will go here
*
*
*/
```

"dependencies" should not include "moment"